

CPE — Parte 2

Vetores de Caracteres (Strings)

Departamento de Engenharia Elétrica – UnB

- 1 Cadeias de caracteres
- 2 Lendo e escrevendo cadeias
- 3 Manipulando cadeias de caracteres

Cadeias de caracteres

- Uma cadeia de caracteres, mais conhecida como *string*, é uma sequência de letras e símbolos, onde os símbolos podem ser espaços em branco, dígitos e vários outros como pontos de exclamação e interrogação, símbolos matemáticos, etc.
- Há duas formas de representar strings em C++
 - Por um vetor de variáveis do tipo `char` e é terminada com o caractere especial `'\0'`. Essa representação é também chamada de **C-strings**.
 - Por uma variável do tipo **`std::string`**, definido na biblioteca padrão do C++. É neste tipo que focaremos nosso estudo.

Declarando uma variável `std::string`

No cabeçalho do programa, deve-se incluir a diretiva para inclusão da biblioteca:

```
#include <string>
```

Exemplo de declaração

```
std::string texto;
```

- Veja que, diferentemente dos outros tipos de vetores, não é necessário definir o tamanho, isto ocorre porque o espaço em memória para variáveis `std::string` é definido em tempo de execução, isto é, o espaço vai sendo alocado à medida que for necessário.

Acesso indexado

Tanto as C-strings como as variáveis `std::string` tem acesso indexado, isto é, podemos referenciar/acessar um determinado caractere da cadeia com o índice entre colchetes:

```
texto[4] = 'f';
```

Lendo uma string do teclado

- Podemos ler uma cadeia caractere a caractere, como faríamos com qualquer outro vetor, mas é mais simples ler a cadeia inteira, ou seja

```
cin >> texto;
```

Exemplo: leia.cpp.

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  int main () {
6
7      string texto;
8
9      /* Interrompe a leitura no primeiro espaco em branco */
10     cin >> texto;
11     cout << texto << endl;
12
13     return 0;
14 }
```

Lendo uma cadeia do teclado

- Infelizmente, a leitura a partir do teclado utilizando `cin >>` lê somente até o primeiro espaço, ou seja, lê somente uma palavra, o que torna o seu uso um pouco restrito.
- Para contornar isso, podemos utilizar a função `getline` para `std::string`

```
getline(cin, texto);
```

Ela faz a leitura até encontrar o caractere de fim de linha (*enter*).

Exemplo: getline.cpp.

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  int main () {
6
7      string texto;
8
9      getline(cin , texto);
10     cout << texto << endl;
11
12     return (0);
13 }
```

Escrevendo uma cadeia na tela

- Podemos escrever uma cadeia na tela caractere a caractere, mas é mais simples escrever de forma direta:

```
cout << texto;
```

Manipulando objetos `string`

A manipulação de variáveis do tipo `std::string` é relativamente fácil. Daí vem um dos benefícios em usar esse tipo de dado da biblioteca padrão do C++ no lugar de um vetor `char` (C-strings).

Considere as variáveis abaixo todas previamente declaradas do tipo `std::string`.

- Para obter o tamanho da string `texto`, basta invocar no código

```
texto.size()
```

Manipulando objetos string

- Para **copiar** a string fonte para a string destino, basta fazer

```
destino = fonte;
```

Manipulando objetos string

- Para **concatenar** a string fonte no fim da string destino, basta fazer

```
destino = destino + fonte;
```

Exemplo: funcoesstring.cpp.

```

1  #include <iostream>
2  #include <string> //cabecalho de strings do C++
3  using namespace std;
4
5  int main () {
6
7      //pode-se declarar e inicializar uma string como aqui emabaixo
8      string feliz = "Eu sou um texto feliz.";
9      string triste = "Eu sou um texto triste.";
10     string indeciso;
11     int l;
12
13     l = feliz.size(); //obtendo tamanho da string
14     cout << "Comprimento do texto feliz: " << l << endl;
15
16     cout << "Valor de indeciso antes de qualquer operacao:\n";
17     cout << indeciso << '\n';
18
19     indeciso = feliz; //copia de strings no C++ pode ser por atribuicao direta
20
21     cout << "Valor de indeciso depois da copia:\n";
22     cout << indeciso << '\n';
23
24     indeciso = indeciso + triste; //concatenacao pode ser feita com o +
25
26     cout << "Valor de indeciso depois da concatenacao:\n";
27     cout << indeciso;
28
29     return 0;
30 }

```

Manipulando objetos `string`

A comparação entre variáveis `std::string` é relativamente fácil, também. Basta usar os operadores relacionais já vistos para tipos numéricos.

`str1 == str2`: dá `true` se `str1` e `str2` forem iguais;

`str1 < str2`: dá `true` se `str1` for menor que `str2`;

`str1 > str2`: dá `true` se `str1` for maior que `str2`;

A comparação pode ser usada, por exemplo, em

- 1 um programa que faça uma busca sequencial por um caractere ou
- 2 em um programa que ordena um **vetor de variáveis `string`**.

Busca sequencial por um caractere numa string

```
1  //Procura um elemento em uma cadeia de caracteres – busca linear.
2  #include <iostream>
3  #include <string>
4  using namespace std;
5
6  int main() {
7      string str;
8      char c;
9      int i;
10     bool achou = false;
11
12     cout << "Digite uma cadeia de caracteres: ";
13     cin >> str;
14     cout << "Digite um caractere: ";
15     cin >> c;
16
17     //laco de busca sequencial
18     for (i = 0; !achou && i<str.size(); i++)
19         if(str[i] == c)
20             achou = true;
21
22     if (achou) //varreu e nao achou
23         cout << "O caractere " << c << " esta presente na string " << str << "\n";
24     else
25         cout << "O caractere " << c << " nao esta presente na string " << str << "\n";
26
27     return 0;
28 }
```


Dilbert

